Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 316 (2015) 83–95

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Dimension Reduction of Large Sparse AND-NOT Network Models

Alan Veliz-Cuba[1]

Department of Mathematics
University of Houston
Houston, Texas, USA
Department of Biochemistry & Cell Biology
Rice University
Houston, Texas, USA

Boris Aguilar[2]

Institute for Systems Biology
Seattle, Washington, USA

Reinhard Laubenbacher[3]

Center for Quantitative Medicine
University of Connecticut Health Center
Farmington, Connecticut, USA
Jackson Laboratory for Genomic Medicine
Farmington, Connecticut, USA

Abstract

In this manuscript we propose and implement a dimension reduction algorithm of AND-NOT networks for the purpose of steady state computation. Our method of network reduction consists in using "steady state approximations" that do not change the number of steady states. The algorithm is designed to work at the wiring diagram level without the need to evaluate or simplify Boolean functions. Also, our implementation of the algorithm takes advantage of the sparsity typical of discrete models of biological systems.
The main features of our reduction algorithm are that it works at the wiring diagram level and it preserves the number of steady states. Furthermore, the steady states of the original network can be recovered from the steady states of the reduced network; thus, all steady states are found. Also, heuristic analysis and simulations show that it runs in polynomial time. We used our results to study AND-NOT network models of gene networks and showed that our algorithm greatly simplifies steady state analysis. Furthermore, our algorithm can handle sparse AND-NOT networks with up to 1,000,000 nodes.

Keywords: Boolean network, steady state, fixed point, network reduction.

[1] Email: alanavc@math.uh.edu
[2] Email: boaguilar@gmail.com
[3] Email: laubenbacher@uchc.edu

http://dx.doi.org/10.1016/j.entcs.2015.06.012

1 Introduction

Boolean networks (BN) have been used in modeling biological networks such as gene regulatory networks [2,12,35,14,31], and provide a good framework for theoretical analysis [21,32,25,13,16,30]. A problem of interest in the analysis of a BN, $f : \{0,1\}^n \to \{0,1\}^n$, is the computation of its steady states (x such that $f(x) = x$). However, this is not a trivial task [36,34,22]. Even comprehensive sampling of the phase space is of limited use, once a model contains 50 or 100 nodes. In this manuscript we focus on the method of network reduction to compute the steady states. A summary of different approaches for steady state computation can be found in [26] and references therein.

Although several reduction algorithms have been proposed [24,17,18,33], it is not clear if such algorithms scale well with the number of nodes. These algorithms are based on using "steady state approximations" to remove nodes in a BN. More precisely, to remove a node i in a BN, $f = (f_1, \dots, f_m) : \{0,1\}^n \to \{0,1\}^n$, one assumes that the i-th variable is at steady state and replaces all instances of the i-th variable by its Boolean function. For example, we can reduce the BN $f(x_1, x_2, x_3) = (\neg x_3, x_1 \vee \neg x_3, x_1 \wedge \neg x_2)$, by making the substitution $x_3 \to f_3 = x_1 \wedge \neg x_2$; then, we obtain the reduced BN $h(x_1, x_2) = (\neg(x_1 \wedge \neg x_2), x_1 \vee \neg(x_1 \wedge \neg x_2))$.

There are two important aspects in the reduction of BNs. One is the representation of the Boolean functions (e.g. Boolean operators, polynomials, binary decision diagrams, truth tables), and the other is the way in which the reduced network is simplified to ensure that the wiring diagram is consistent with the Boolean functions (e.g. Boolean algebra, polynomial algebra, substitution). It is in these two aspects where algorithms can stop being scalable. For example, although polynomial algebra makes the manipulation of Boolean functions very systematic, the polynomial representation of Boolean functions can be large. For instance, storing $x_1 \vee x_2 \vee \ldots \vee x_k$ and $\neg x_1 \wedge \neg x_2 \wedge \ldots \wedge \neg x_k$ in polynomial form grows exponentially with respect to k. On the other hand, although using Boolean operators can be more intuitive and efficient at representing Boolean functions, their simplification also grows exponentially with respect to the number of variables.

The reduction algorithm in this paper is tailored specifically to the computation of steady states of AND-NOT networks and takes advantage of the sparsity typical of gene regulatory networks. AND-NOT networks are BNs where the functions are of the form $y_{i_1} \wedge y_{i_2} \wedge \cdots \wedge y_{i_r}$ where $y_{i_j} \in \{x_{i_j}, \neg x_{i_j}\}$. We focus on AND-NOT networks because they have been shown to be "general enough" for modeling and "simple enough" for theoretical analysis [27,29,6]. Also, synthetic AND-NOT gene networks can be designed by coupling synthetic AND gates [20] and negative regulation. AND-NOT functions are a particular case of nested canalizing functions, which have been proposed as a class of BNs for modeling biological systems [9,10,8,7,15].

Our dimension reduction algorithm for AND-NOT networks has two important properties: First, it preserves all steady state information; more precisely, there is a one-to-one correspondence between the steady states of the original and reduced network. Second, it works at the wiring diagram level, which makes it very efficient

for large sparse networks. Also, the steady states of the original network can be recovered from the steady states of the reduced network. As in previous reduction methods, the main idea of our algorithm is that one can use steady state approximations without changing the number of steady states; however, there are some key differences which will explained in Section 3. First, the only reduction steps that are allowed are those that result in a reduced AND-NOT network. Second, since we are using AND-NOT networks only, we can make additional reductions that cannot be done with other networks. Since AND-NOT networks are completely determined by their wiring diagrams, we can store AND-NOT networks efficiently using their wiring diagrams (avoiding the problem that the polynomial representation) and we can state all reduction steps and simplification of the reduced network at the wiring diagram level (avoiding the problem that the Boolean representation has).

2 Preliminaries

Definition 2.1 An *AND-NOT function* is a Boolean function, $b : \{0,1\}^n \to \{0,1\}$, that can be written as $b = b(x_1, \dots, x_n) = \bigwedge_{i\in P} x_i \wedge \bigwedge_{i\in N} \neg x_i$; where $P \cap N = \{\ \}$, "$\wedge$" is the AND operator, and "$\neg$" is the NOT operator. If $P = N = \{\ \}$, then b is the constant 1 (by convention $\bigwedge_{i\in\{\ \}} x_i = \bigwedge_{i\in\{\ \}} \neg x_i = 1$).

Definition 2.2 An *AND-NOT network* is a BN, $f = (f_1, \dots, f_n) : \{0,1\}^n \to \{0,1\}^n$, such that for all j, f_j is an AND-NOT function or the constant function 0. AND-NOT networks are also called *signed conjunctive networks*. Consider $f_j = \bigwedge_{i\in P} x_i \wedge \bigwedge_{i\in N} \neg x_i$. If $i \in P$ ($i \in N$, respectively) we say that i or x_i is a *positive* (*negative*) regulator of j or that it is an activator (repressor).

Example 2.3 The BN $f : \{0,1\}^6 \to \{0,1\}^6$ given by

$$f_1 = x_2 \wedge x_4 \wedge \neg x_5, \qquad f_2 = \neg x_3 \wedge \neg x_5 \wedge x_6, \qquad f_3 = 0,$$
$$f_4 = \neg x_1 \wedge \neg x_5 \wedge x_6, \qquad f_5 = x_6, \qquad f_6 = 1,$$

is an AND-NOT network (e.g. f_1 can be written as, $f_1 = \bigwedge_{i\in\{2,4\}} x_i \wedge \bigwedge_{i\in\{5\}} \neg x_i$).

Definition 2.4 We say that x is a *steady state* or *fixed point* of a BN f if $f(x) = x$; that is, if for all $i = 1, \dots, n$ we have that $f_i(x) = x_i$. For example, it is easy to check that 000011 is a steady state of the AND-NOT network in Example 2.3.

Definition 2.5 The *extended wiring diagram* of an AND-NOT network is defined as a signed directed graph $G = (V_G, E_G)$ with vertices $V_G = \{0, 1, \dots, n\}$ (or $\{0, x_1, \dots, x_n\}$) and edges E_G given as follows: $(i, j, +) \in E_G$ ($(i, j, -) \in E_G$, respectively) if x_i is a positive (negative, respectively) regulator of f_j. If $f_j = 0$, then $(0, j, +) \in E_G$. Positive edges are denoted by —▶ and negative edges by —●. We will refer to the extend wiring diagram as simply *wiring diagram*. Note that an AND-NOT network is completely determined by its wiring diagram. For example, the wiring diagram of the AND-NOT network in Example 2.3 is shown in Figure 1.

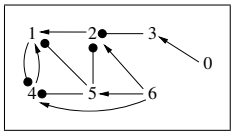

Fig. 1. Wiring diagram of the AND-NOT network in Example 2.3.

3 Reduction of AND-NOT Networks

3.1 Reduction Steps and Algorithm

As mentioned in the Introduction, the idea is to assume that nodes are in steady state and remove them from the network by replacing the variable by the corresponding AND-NOT function. At the wiring diagram level, the idea is to remove nodes and insert edges so that the sign of the edges are "consistent". For example, a path $i \dashrightarrow j \dashv\!\bullet k$ should become $i \dashv\!\bullet k$ after removing node j; and $i \dashv\!\bullet j \dashv\!\bullet k$ should become $i \dashrightarrow k$ after removing node j. The actual rules for doing this depend on the properties of the node being removed and the incoming and outgoing edges.

Figure 2 shows the steps at the wiring diagram level. We claim that each of these reduction steps do not change the number of steady states and that the one-to-one correspondence is algorithmic. The proofs follow directly from basic properties of Boolean algebra, so we only describe the effect on the wiring diagram and give the idea behind each reduction step. We note that all nodes may have additional incoming or outgoing edges not drawn unless stated otherwise.

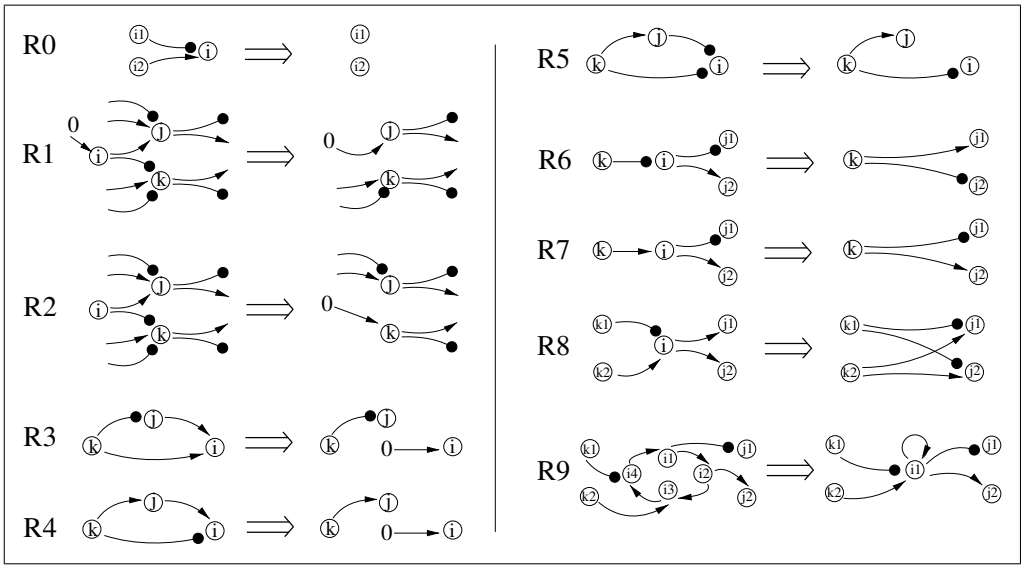

Fig. 2. Reduction steps (before and after). Circles denote nodes. All nodes can have more inputs/outputs not drawn in the figure with the following exceptions: node i in $R0$ does not have any outgoing edges; node i in $R2$ does not have any incoming edge; node i in $R5$ does not have any other incoming edge; node i in $R6$ and $R7$ does not have any other incoming edge; node i in $R8$ has positive outgoing edges only.

- **Reduction Step R0.** Here node i does not have any outgoing edges, so this node does not contribute to the number of steady states and can be removed.

  Note that given a steady state of the reduced AND-NOT network, the steady

state of the original network can be found simply by inserting (in the i-th entry) $x_i = f_i$. Note that this reduction step is also valid for general BNs.

- **Reduction Step R1.** Here i has one incoming edge from node 0. Then, we have $f_i = 0$; and we remove node i by replacing x_i with $f_i = 0$. After reduction, node j will only have one incoming edge (from node 0).

  For example, if $i \longrightarrow j$, then $f_j = x_i \wedge w_j$ for some AND-NOT function w_j. By replacing x_i with 0 we obtain $f_j = 0 \wedge w_j = 0$; that is, we add the edge $0 \longrightarrow j$ and remove all other incoming edges of j. On the other hand, if $i \multimap k$, then $f_k = \neg x_i \wedge w_k$ for some AND-NOT function w_k. By replacing x_i with 0 we obtain $f_j = \neg 0 \wedge w_k = w_k$; that is, the edge $i \multimap k$ is removed and all other edges towards k remain present. Note that given a steady state of the reduced AND-NOT network, the steady state of the original network can be found simply by inserting (in the i-th entry) $x_i = 0$. We note that a similar reduction step is valid for general BNs.

- **Reduction Step R2.** Here i does not have any incoming edge. Then, we have $f_i = 1$; and we remove node i by replacing x_i with $f_i = 1$. After reduction, node k will only have one incoming edge (from node 0).

  For example, if $i \longrightarrow j$, then $f_j = x_i \wedge w_j$ for some AND-NOT function w_j. By replacing x_i with 1 we obtain $f_j = 1 \wedge w_j = w_j$; that is, the edge $i \multimap j$ is removed and all other edges towards j remain present. On the other hand, if $i \multimap k$, then $f_k = \neg x_i \wedge w_k$ for some AND-NOT function w_k. By replacing x_i with 1 we obtain $f_j = \neg 1 \wedge w_k = 0$; that is, we add the edge $0 \longrightarrow k$ and remove all other incoming edges of k. Note that given a steady state of the reduced AND-NOT network, the steady state of the original network can be found simply by inserting $x_i = 1$. We note that a similar reduction step is valid for general BNs.

- **Reduction Steps R3, R4.** Here we have edges from k to j and from k to i of opposite signs, and a positive edge from j to i (other edges may be present as well). After reduction, all incoming edges of i are removed, and replaced by a single edge from node 0.

  For $R3$ we have $f_i = x_j \wedge x_k \wedge w_i$ for some AND-NOT function w_i, and a node j with Boolean function $f_j = \neg x_k \wedge w_j$ for some AND-NOT function w_j. If we are at a steady state, then we have two cases, either $x_k = 0$ or $x_k = 1$. If $x_k = 0$, then $x_i = f_i = x_j \wedge 0 \wedge w_i = 0$. If $x_k = 1$, then $x_j = f_j = \neg 1 \wedge w_j = 0$, and then $x_i = 0 \wedge x_k \wedge w_i = 0$. In either case $x_i = 0$, so by assuming that $f_i = 0$ we are not changing the steady states of the AND-NOT network. That is, we add the edge $0 \longrightarrow i$ and remove all other incoming edges of i. The reduction step $R4$ is analogous. It is important to mention that these reduction steps are not valid for general BNs.

- **Reduction Step R5.** Node i has two incoming edges only, from k and j; there is also a positive edge from k to j (other edges may be present as well). After reduction, the edge from j to i is removed.

  We have that $f_i = \neg x_j \wedge \neg x_k$ and $f_j = x_k \wedge w_j$ for some AND-NOT function w_j. If we are at a steady state then we have two cases, either $x_k = 0$ or $x_k = 1$.

If $x_k = 0$, then $x_j = f_j = 0 \wedge w_j = 0$ and $x_i = f_i = \neg 0 \wedge \neg 0 = 1$. If $x_k = 1$, then $x_i = f_i = \neg x_j \wedge \neg 1 = 0$. In either case we have $x_i = \neg x_k$, so by assuming that $f_i = \neg x_k$ we are not changing the steady states. It is important to mention that this reduction step is not valid for general BNs.

- **Reduction Step R6, R7.** Here we have a node i with a single incoming edge. After reduction, paths of the form $k \longrightarrow i \multimap j_1$ and $k \longrightarrow i \longrightarrow j_2$ are replaced by edges $k \multimap j_1$ and $k \longrightarrow j_2$, respectively. Then, node i, its incoming edge, and its outgoing edges are removed.

  For $R6$ we have that $f_i = \neg x_k$; and we remove node i by replacing x_i with $f_i = \neg x_k$. For example, if $f_{j_1} = \neg x_i \wedge w_{j_1}$ for some AND-NOT function w_{j_1}, then we obtain $f_{j_1} = \neg\neg x_k \wedge w_{j_1} = x_k \wedge w_{j_1}$, which is an AND-NOT function. If $f_{j_2} = x_i \wedge w_{j_2}$ for some AND-NOT function w_{j_2}, then we obtain $f_{j_2} = \neg x_k \wedge w_{j_2}$, which is an AND-NOT function as well. Note that given a steady state of the reduced AND-NOT network, the steady state of the original network can be found simply by inserting $x_i = \neg x_k$. The reduction step $R7$ is analogous. The reduction is no longer valid if i has more incoming edges (the reduced network would not be an AND-NOT network). We note that a similar reduction step is valid for general BNs.

- **Reduction Step R8.** Here we have that all outgoing edges of i are positive. After reduction, paths of the form $k_1 \multimap i \longrightarrow j$ and $k_2 \longrightarrow i \longrightarrow j$ are replaced by edges $k_1 \multimap j$ and $k_2 \longrightarrow j$, respectively. Then, node i, its incoming edges, and its outgoing edges are removed.

  We remove node i by replacing x_i with f_i. For example, if $f_i = \neg x_{k_1} \wedge x_{k_2}$ and $f_{j_1} = x_i \wedge w_j$ for some AND-NOT function w_{j_1}, then we obtain $f_{j_1} = \neg x_{k_1} \wedge x_{k_2} \wedge w_{j_1}$, which is an AND-NOT function. Note that given a steady state of the reduced AND-NOT network, the steady state of the original network can be found simply by inserting $x_i = \neg x_{k_1} \wedge x_{k_2}$. It is important to mention that the reduction is no longer valid if i has any negative outgoing edge. We note that a similar reduction step is valid for general BNs.

- **Reduction Step R9.** Here we have a circuit with positive edges only $i_1 \longrightarrow i_2 \ldots \longrightarrow i_r \longrightarrow i_1$. After reduction, edges of the form $k_1 \multimap i_s$, $k_2 \longrightarrow i_s$, $i_s \multimap j_2$, and $i_s \longrightarrow j_2$ are replaced by edges $k_1 \multimap i_1$, $k_2 \longrightarrow i_1$, $i_1 \multimap j_2$, and $i_1 \longrightarrow j_2$. Then, nodes i_s for $s \geq 2$ are removed, as well as their incoming and outgoing edges. It is important to mention that this reduction step is not valid for general BNs.

  Figure 2 shows the effect of the reduction on a circuit with 4 nodes. If we are at a steady state, we have two cases, either $x_{i_1} = 0$ or $x_{i_1} = 1$. If $x_{i_1} = 0$, then it follows that $x_{i_2} = 0$ and working forward we obtain that $x_{i_1} = x_{i_2} = \ldots = 0$. Similarly, if $x_{i_1} = 1$, we obtain that $x_{i_1} = x_{i_2} = \ldots = 1$. Thus, by collapsing this circuit into a single node we do not change the number of steady states. Note that given a steady state of the reduced AND-NOT network, the steady state of the original network can be found simply by inserting $(x_{i_2}, x_{i_3}, x_{i_4}, \ldots) = (x_{i_1}, x_{i_1}, x_{i_1}, \ldots)$. Note that this reduction step is no longer valid if one of the edges in the circuit is negative. It is important to mention that this reduction step is not valid for general BNs.

It is important to mention that reduction steps $R0 - R8$ cover the possible reductions where we only need to look at incoming and outgoing edges of a node i. Other reduction steps could be considered by looking upstream and downstream of a node; for example, one can generalize $R3$ and $R4$ to include longer feedforward loops (e.g. $i_1 \longrightarrow i_2 \longrightarrow \ldots \longrightarrow i_r$, $i_1 \multimap i_r$). However, their detection becomes computationally expensive. Nevertheless, by using the reduction steps iteratively, one can perform further reductions. For example, linear chains can be reduced using steps $R6$ and $R7$ repeatedly. Reduction step $R9$ is fast even for a large number of nodes because such circuits can be detected in linear time with respect to the number of edges [23].

The actual algorithm is given below. The idea is to iteratively apply the reduction steps until the network is no longer reducible (every time a reduction step is used, new reducible nodes may appear). Note that there are many orders in which one can apply the reduction steps, and in some cases they can result in different reduced networks (with the same number of steady states). Based on the performance of preliminary simulations, the order given below was chosen.

Reduction Algorithm.
Input: AND-NOT network G.
Output: Reduced AND-NOT network.

(i) Use $R0$ to remove terminal nodes.
(ii) Let $Z = \{j : (0, j, +) \in E_G \text{ or } I_j(G) = \{\}\}$. If $Z = \{\}$, then go to (v).
(iii) Use $R1, R2$ to remove from G the nodes in Z.
(iv) Go to (i).
(v) Use $R3, R4$ to find new nodes with input 0.
(vi) If nodes were found in previous step, then go to (i).
(vii) Use $R5$ to remove edges.
(viii) If there are nodes with a single incoming edge, then use $R6, R7$ and go to (i).
(ix) Find nodes with positive outgoing edges only.
(x) If nodes were found in previous step, then use $R8$ and go to (i).
(xi) Find circuits with positive edges only (only use this step once).
(xii) If circuits were found in previous step, then use $R9$ and go to (i).

Note that step $R9$ is used only once in the algorithm because none of the other steps create extra circuits that contain positive edges only (although other types of circuits may be created).

Once we have the reduced AND-NOT network we need to compute its steady states. Since for sparse networks the reduced network will be much smaller (see next two sections), standard techniques can now be used. It is important to mention that at each step the steady states of a reduced network can be recovered from the steady states of the reduced network by substitution. Then, at the end of the reduction algorithm we have a chain of equations that can be represented as an acyclic graph. Thus, the result of our reduction algorithm is not only a reduced network with the

same number of steady states, but it also gives an acyclic graph that encodes the backwards substitution needed to recover the steady states of the original network.

3.2 Implementation and Computational Complexity

We preliminarily implemented our algorithm in C++ and used the Boost Graph Library to manipulate graphs (code available upon request). We stored the one-to-one correspondence as an acyclic graph so that once the steady states of the reduced network are computed, one simply uses backward substitution to recover the steady states of the original network. The steady states of the reduced AND-NOT network are computed by exhaustive search.

Example 3.1 Consider the AND-NOT network given by:

$$f_1 = x_4, \qquad f_2 = x_1 \wedge \neg x_3 \wedge x_4, \qquad f_3 = 0,$$
$$f_4 = x_1, \qquad f_5 = \neg x_2 \wedge x_4 \wedge x_6, \qquad f_6 = 1.$$

The wiring diagram of this AND-NOT network is in Figure 3 (left). After using our reduction algorithm we obtain the network $h(x_4) = x_4$. Also, we obtain the equations that give the bijection between steady states of the reduced and original network (Figure 3, right):

$$x_3 = 0, \qquad x_6 = 1, \qquad x_1 = x_4, \qquad x_2 = x_4, \qquad x_5 = \neg x_2 \wedge x_4 \wedge x_6.$$

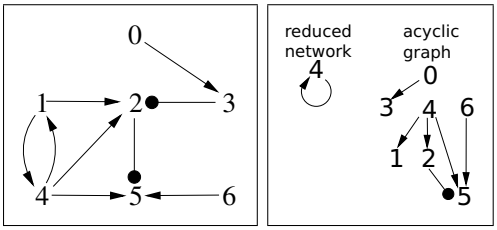

Fig. 3. Left: Wiring diagram of the AND-NOT network in Example 3.1. Right: The reduced network and the acyclic graph given by the reduction algorithm.

That is, for each steady state of the reduced network h, we can recover the steady states of the original network by using backwards substitution. We easily obtain the steady states of h: $x_4 = 0, 1$. For $x_4 = 0$ we obtain $x_3 = 0$, $x_6 = 1$, $x_2 = x_4 = 0$, $x_1 = x_4 = 0$, $x_5 = x_1 \wedge \neg x_2 \wedge x_6 = 0 \wedge \neg 0 \wedge 1 = 0$; that is, $x = 000001$. For $x_4 = 1$ we obtain $x_3 = 0$, $x_6 = 1$, $x_2 = x_4 = 1$, $x_1 = x_4 = 1$, $x_5 = x_1 \wedge \neg x_2 \wedge x_6 = 1 \wedge \neg 1 \wedge 1 = 0$; that is, $x = 110101$.

Since it is not known the average number of times each pattern in Figure 2 appears in a random AND-NOT network, it is difficult to predict the exact computational complexity of our algorithm. However, we present a heuristic estimation as follows. Let n be the number of nodes and e the number of edges; we denote with T the computational complexity of our reduction algorithm including the backwards substitution step. In the worst case scenario steps (xi) and (xii) will have to be done at the beginning, this contributes $O(n + e)$ to T [23]. Each detection of an individual pattern and reduction step from (i) to (x) takes constant time for each node; then, each one of these steps of the algorithm (not counting the "go to" statements) contributes $O(n)$ to T; and, since we have to repeat this at most n times

(counting the "go to" statements), we obtain that steps (i)-(x) contribute $O(n^2)$ to T. Thus, steps (i)-(xii) contribute $O(n+e+n^2) = O(n^2)$ to T. Finally, note that the backwards substitution part contributes $O(n^2)$ to T. Thus $T = O(n^2)$.

Although our reduction algorithm seems to run in polynomial time, the computation of steady states of the reduced network can be much more computationally expensive for general AND-NOT networks. However, for sparse networks the reduced AND-NOT network can be in practice several orders of magnitude smaller than the original network. Indeed, for the Boolean models and random networks that we study in the next section, the size of the reduced networks, m, had the same order of magnitude as $\ln(n)$. This is important because if $m = O(\ln(n))$, then the time to compute the steady states of the reduced network is $O(n^l)$ for some $l > 0$ and thus the total time of steady state computation (reduction, computation of steady states of reduced network, and backwards substitution) is $O(n^2 + n^l) = O(n^k)$, where $k = \max(2, l)$.

4 Application

In this section we apply our reduction algorithm to three published networks and large random networks, and demonstrate that it can result in a significant reduction of the network's dimension. We denote two negative (positive) edges between i and j by a bidirectional negative (positive) edge, •—• (◀—▶); if the edges have different signs we denote them by •—▶.

4.1 Boolean Models

We considered AND-NOT networks based on Boolean models of Th-cell differentiation [14], ERBB2 activation [19], and T-cell receptor [11]. The original networks were not AND-NOT networks, so we constructed the AND-NOT representation for each original Boolean model using the algorithm given in [27], which preserves the steady states (up to a bijection). The wiring diagrams of the three AND-NOT networks are shown in Figure 4. The number of nodes in these AND-NOT networks was 26, 24, and 43, respectively.

Our algorithm took less than 3ms to reduce each AND-NOT network. The reduced networks are shown in Figure 4 (insets) and are much smaller.

4.2 Random AND-NOT networks

In this section we show that our algorithm works very well for large sparse AND-NOT networks. We run our implementation of the algorithm on a Linux system using one 2.40GHz CPU core. To mimic wiring diagrams of gene regulatory networks, we considered random AND-NOT networks with wiring diagrams where the in-degree followed a power law distribution [5,3,1] with no constant nodes. Since the parameter γ in the power law distribution is usually between 2 and 3 for biochemical networks [3,1], we considered the parameters $\gamma = 2.0, 2.2, 2.4, 2.6, 2.8, 3.0$. We analyzed about 100000 AND-NOT networks. The summary of the analysis for

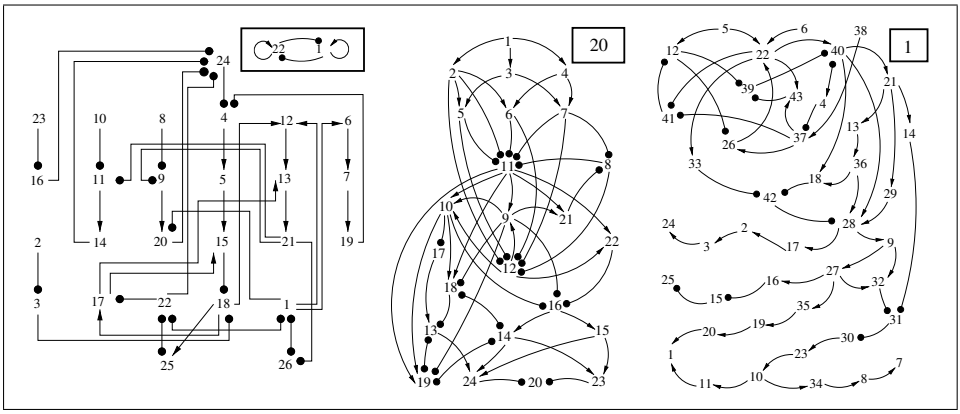

Fig. 4. AND-NOT models of Th-cell differentiation (left), ERBB2 activation (center), and T-cell receptor (right). The insets show the reduced networks.

n	$\gamma = 2.0$	$\gamma = 3.0$
10^2	$\mu_t = .002838, \sigma_t = .000603$	$\mu_t = .002398, \sigma_t = .000710$
10^3	$\mu_t = .018034, \sigma_t = .003995$	$\mu_t = .009058, \sigma_t = .001744$
10^4	$\mu_t = .597764, \sigma_t = .242391$	$\mu_t = .092602, \sigma_t = .015238$
10^5	$\mu_t = 246.241, \sigma_t = 147.727$	$\mu_t = 2.13928, \sigma_t = .493422$
10^6	$\mu_t = 13661.1, \sigma_t = 9129.92$	$\mu_t = 31.0311, \sigma_t = 5.36648$
Best fit of $t = cn^k$	$c \approx 10^{-8}, k \approx 2.025$	$c \approx 2 \times 10^{-6}, k \approx 1.197$

Table 1
Timing, t, for the reduction algorithm (in seconds). The mean and standard deviation of t are denoted by μ_t and σ_t, respectively. Last row: best fit polynomial $t = cn^k$.

$\gamma = 2$ and $\gamma = 3$ is Table 4.2. Figure 5 shows the plots of time (t) v.s. the size of the network (n) for $\gamma = 2.0, 2.2, 2.4, 2.6, 2.8, 3.0$ in a log-log scale. These timings include the timing of the reduction steps and the timing of steady state computation, although the latter turned out to be negligible. More precisely, the number of nodes of the reduced AND-NOT networks, m, was very small with an average of $\mu_m = 2.6$, ranging from $m = 0$ to $m = 19$. Since these numbers are small, exhaustive search was sufficient to compute the steady states of the reduced networks. It is important to mention that if the reduced AND-NOT network is too large to handle by exhaustive search, one can use additional tools such as polynomial algebra [28,4], but as mentioned before, this was not necessary for our simulations.

We can see in Figure 5 that our reduction algorithm scales well with the number of nodes. Furthermore, our algorithm can reduce networks with up to 1,000,000 nodes. We also see that for very sparse networks (i.e. large values of γ) our algorithm scales very well. As sparsity is lost (i.e. as γ decreases), our algorithm becomes less and less scalable; however, as mentioned before, the value of γ for biochemical networks is usually between 2 and 3 for which our algorithm performs well. Also, the timings appear polynomial (linear on a log-log scale), especially for large γ and n. The best fit polynomial of the form $t = cn^k$ for $\gamma = 2$ was given by $c \approx 10^{-8}$ and $k \approx 2.025$; and for $\gamma = 3$ was given by $c \approx 2 \times 10^{-6}$ and $k \approx 1.197$. Note: Although using the timings at $n = 10^2$ for the estimation of c and k would give a smaller value of k, we did not use them because the linear relationship between $\log(t)$ and

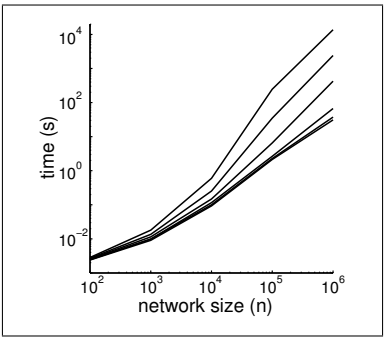

Fig. 5. Average timing (in seconds) v.s. number of nodes for different values of γ. From top to bottom: $\gamma = 2.0, 2.2, \ldots, 3.0$.

$\log(n)$ seemed to start at $n = 10^3$.

5 Discussion

Since the problem of analyzing BNs is hard for large networks, many reduction algorithms have been proposed [24,17,18,33]. However, it is not clear if such algorithms scale well with the size of the network. In order to optimize reduction algorithms, it is necessary to focus on specific families of BNs. The family of AND-NOT networks has been proposed as a special family simple enough for theoretical analysis, but general enough for modeling [27,29,6]. Thus, we propose an algorithm for network reduction for the family of AND-NOT networks. A key property of our algorithm is that it preserves steady states, so it can be very useful in steady state analysis.

We applied our algorithm to three AND-NOT network models, and it performed very well; the reduced networks were several orders of magnitude smaller than the original network and the reduction took less than 3ms. Using random AND-NOT networks, we showed that our algorithm scales well with the number of nodes and can handle large sparse AND-NOT networks with up to 1,000,000 nodes. To the best of our knowledge, no other algorithm can handle AND-NOT networks or any other class of (nonlinear) BNs of this size.

References

[1] Albert, R. *Scale-free networks in cell biology.* Journal of Cell Science, **118(21)** (2005), 4947–4957.

[2] Albert, R. and H. Othmer. *The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in* Drosophila melanogaster. J. Theor. Biol., **223** (2003), 1–18.

[3] Aldana, M. *Boolean dynamics of networks with scale-free topology.* Physica D: Nonlinear Phenomena, **185(1)** (2003), 45–66.

[4] Hinkelmann, F., M. Brandon, B. Guang, R. McNeill, G. Blekherman, A. Veliz-Cuba and R. Laubenbacher. *ADAM: Analysis of Discrete Models of Biological Systems Using Computer Algebra.* BMC Bioinformatics, **12** (2011), 295.

[5] Huynen, M. and E. van Nimwegen. *The frequency distribution of gene family sizes in complete genomes.* Molecular Biology and Evolution, **15(5)** (1998), 583–589.

[6] Jarrah, A., R. Laubenbacher, and A. Veliz-Cuba. *The dynamics of conjunctive and disjunctive Boolean network models.* Bull. Math. Bio., **72(6)** (2010), 1425–1447.

[7] Jarrah, A., B. Raposa, and R. Laubenbacher. *Nested canalyzing, unate cascade, and polynomial functions.* Physica D: Nonlinear Phenomena, **233(2)** (2007), 167–174.

[8] Just, W., I. Shmulevich, and J. Konvalina. *The number and probability of canalizing functions.* Physica D: Nonlinear Phenomena, **197(3-4)** (2004), 211–221.

[9] Kauffman, S., C. Peterson, B. Samuelsson, and C. Troein. *Random Boolean network models and the yeast transcriptional network.* Proc. Natl. Acad. Sci. U.S.A., **100(25)** (2003), 14796–14799.

[10] Kauffman, S., C. Peterson, B. Samuelsson, and C. Troein. *Genetic networks with canalyzing Boolean rules are always stable.* Proc. Natl. Acad. Sci. U.S.A., **101(49)** (2004), 17102–17107.

[11] Klamt, S., J. Saez-Rodriguez, J. Lindquist, L. Simeoni, and E. Gilles. *A methodology for the structural and functional analysis of signaling and regulatory networks.* BMC Bioinformatics, **7(1)** (2006), 56.

[12] Li, F., T. Long, Y. Lu, Q. Ouyang, and C. Tang. *The yeast cell-cycle network is robustly designed.* Proc. Natl. Acad. Sci. U.S.A., **101(14)** (2004), 4781–4786.

[13] Li, W., L. Cui, and M. Ng. *On computation of the steady-state probability distribution of probabilistic Boolean networks with gene perturbation.* Journal of Computational and Applied Mathematics, **236(16)** (2012), 4067–4081.

[14] Mendoza, L. and I. Xenarios. *A method for the generation of standardized qualitative dynamical systems of regulatory networks.* Theoretical Biology and Medical Modelling, **3(1)** 2006, 13.

[15] Murrugarra, D. and R. Laubenbacher. *The number of multistate nested canalyzing functions.* Physica D: Nonlinear Phenomena, **241(10)** (2012), 929–938.

[16] Murrugarra, D., A. Veliz-Cuba, B. Aguilar, S. Arat and R. Laubenbacher. *Modeling Stochasticity and Variability in Gene Regulatory Networks.* EURASIP Journal on Bioinformatics and Systems Biology, **2012** (2012), 5.

[17] Naldi, A., E. Remy, D. Thieffry, and C. Chaouiya. *A reduction of logical regulatory graphs preserving essential dynamical properties.* In Pierpaolo Degano and Roberto Gorrieri, editors, *Computational Methods in Systems Biology*, "Lecture Notes in Computer Science", Springer Berlin, Heidelberg, **5688** (2009), 266–280.

[18] Saadatpour, A., I. Albert, and R. Albert. *Attractor analysis of asynchronous Boolean models of signal transduction networks.* Journal of Theoretical Biology, **266(4)** (2010), 641–656.

[19] Sahin, O., H. Frohlich, C. Lobke, U. Korf, S. Burmester, M. Majety, J. Mattern, I. Schupp, C. Chaouiya, D. Thieffry, A. Poustka, S. Wiemann, T. Beissbarth, and D. Arlt. *Modeling erbb receptor-regulated g1/s transition to find novel targets for de novo trastuzumab resistance.* BMC Systems Biology, **3(1)** (2009), 1.

[20] Shis, D. and M. Bennett. *Library of synthetic transcriptional AND gates built with split T7 RNA polymerase mutants.* Proc. Natl. Acad. Sci. U.S.A., **110(13)** (2013), 5028–5033.

[21] Sontag, E., A. Veliz-Cuba, R. Laubenbacher and A. Jarrah. *The Effect of Negative Feedback Loops on the Dynamics of Boolean Networks.* Biophysical Journal, **95** (2008), 518-526.

[22] Tamura, T. and T. Akutsu. *Detecting a Singleton Attractor in a Boolean Network Utilizing SAT Algorithms.* IEICE Transactions on Fundamental Electronics, Communications and Computer Sciences, **E92-A(2)** (2009), 493–501.

[23] Tarjan R. *Depth-First Search and Linear Graph Algorithms.* SIAM Journal on Computing, **1(2)** (1972), 146–160.

[24] Veliz-Cuba, A. *Reduction of Boolean network models.* Journal of Theoretical Biology, **289** (2011), 167–172.

[25] Veliz-Cuba A. *An Algebraic Approach to Reverse Engineering Finite Dynamical Systems Arising from Biology.* SIAM Journal on Applied Dynamical Systems, **11(1)** (2012), 31-48.

[26] Veliz-Cuba, A., B. Aguilar, F. Hinkelmann, and R. Laubenbacher. *Steady state analysis of Boolean molecular network models via model reduction and computational algebra.* BMC Bioinformatics, **15** (2014), 221.

[27] Veliz-Cuba, A., K. Buschur, R. Hamershock, A. Kniss, E. Wolff, and R. Laubenbacher. *AND-NOT logic framework for steady state analysis of Boolean network models.* Applied Mathematics and Information Sciences, **7(4)** (2013), 1263-1274.

[28] Veliz-Cuba, A., A. Jarrah, and R. Laubenbacher. *The Polynomial Algebra of Discrete Models in Systems Biology.* Bioinformatics, **26** (2010), 1637–1643.

[29] Veliz-Cuba, A. and R. Laubenbacher. *On the computation of fixed points in Boolean networks.* Journal of Applied Mathematics and Computing, **39(1-2)** (2011), 145–153.

[30] Veliz-Cuba, A., D. Murrugarra and R. Laubenbacher. *Structure and Dynamics of Acyclic Networks.* Discrete Event Dynamic Systems, accepted.

[31] Veliz-Cuba, A. and B. Stigler. *Boolean models can explain bistability in the lac operon.* J. Comput. Biol., **18(6)** (2011), 783–794.

[32] Xu, W., W. Ching, S. Zhang, W. Li, and X. Chen. *A matrix perturbation method for computing the steady-state probability distributions of probabilistic Boolean networks with gene perturbations.* Journal of Computational and Applied Mathematics, **235(8)** (2011), 2242–2251.

[33] Zañudo, J. and R. Albert. *An effective network reduction approach to find the dynamical repertoire of discrete dynamic networks.* Chaos: An Interdisciplinary Journal of Nonlinear Sciences, **23(2)** (2013), 025111.

[34] Zhang, S., M. Hayashida, T. Akutsu, W. Ching, M. Ng. *Algorithms for finding small attractors in Boolean networks.* EURASIP J. Bioinformatics Syst. Biol., **2007** (2007), 20180.

[35] Zhang, Y., M. Qian, Q. Ouyang, M. Deng, F. Li, and C. Tang. *Stochastic model of yeast cell-cycle network.* Physica D: Nonlinear Phenomena, **219(1)** (2006):35–39.

[36] Zhao, Q. *A remark on scalar equations for synchronous Boolean networks with biological applications by C. Farrow, J. Heidel, J. Maloney, and J. Rogers.* IEEE Transactions on Neural Networks, **16(6)** (2005), 1715–1716.